雲林科技大學

網路最佳化: Non-blocking socket 多人聊天室

學生:薛華慶

學號: M10917055

目錄

程式流程

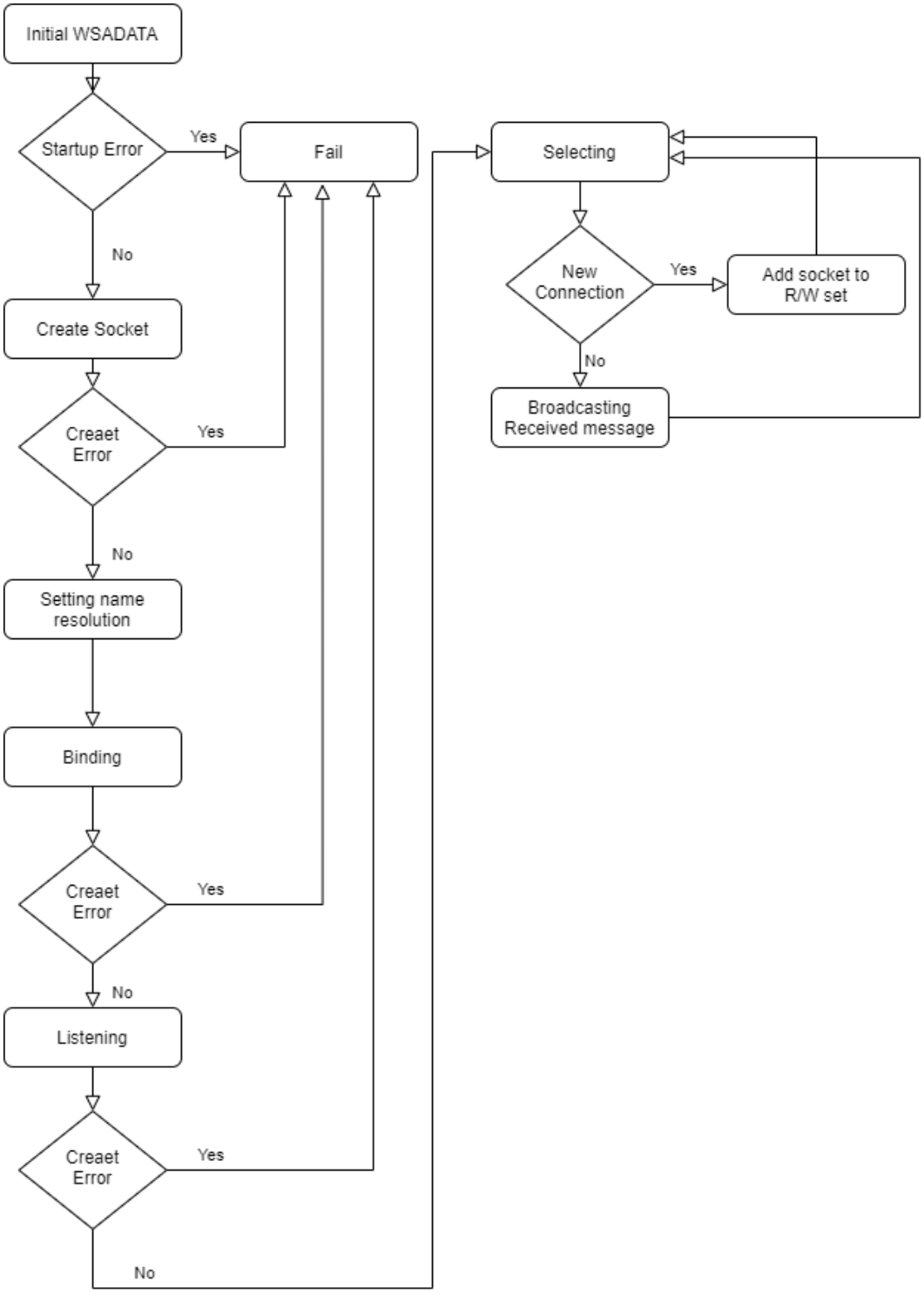

圖 1 Server program flow

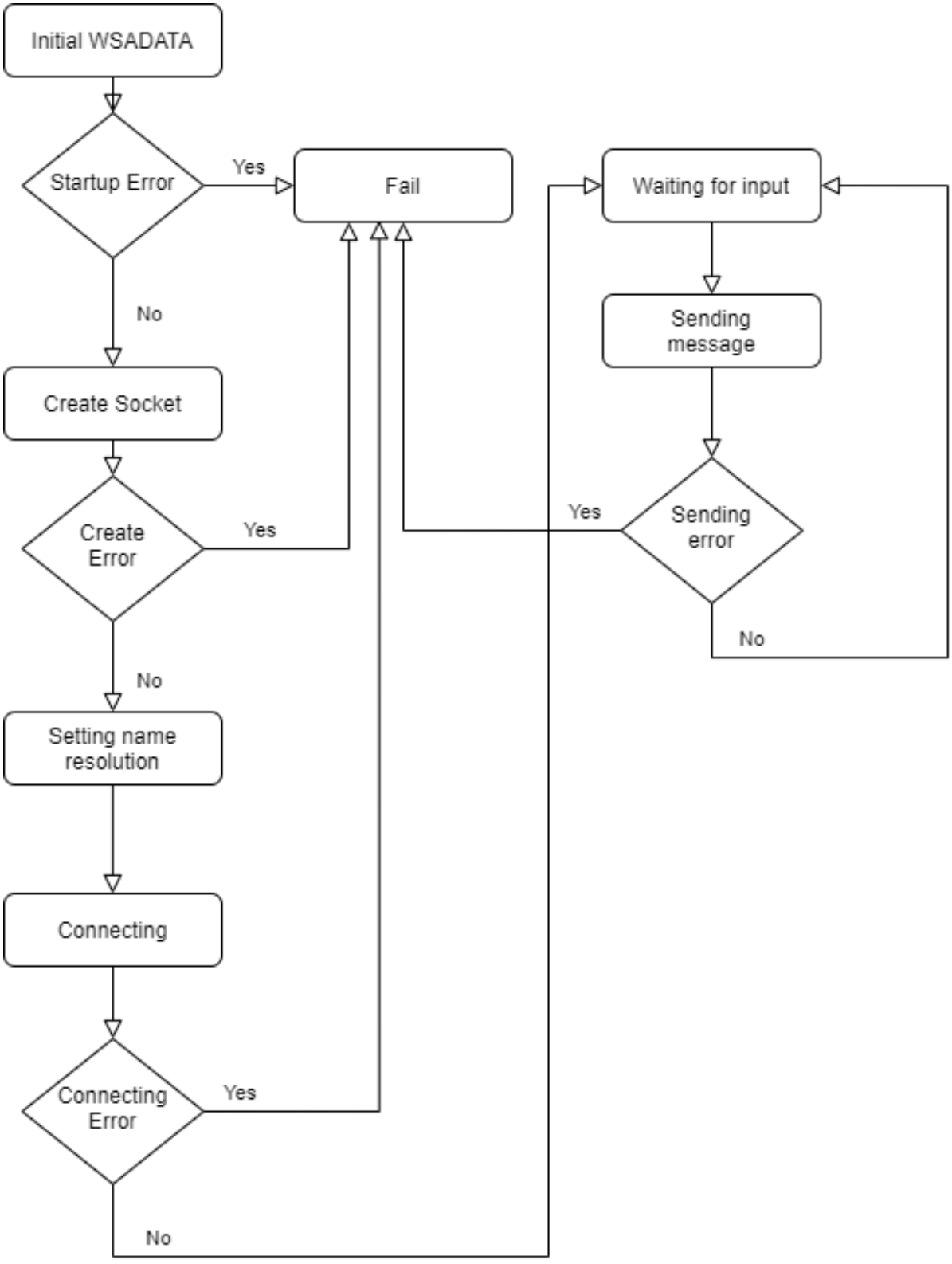

圖 2 Client Writer flow

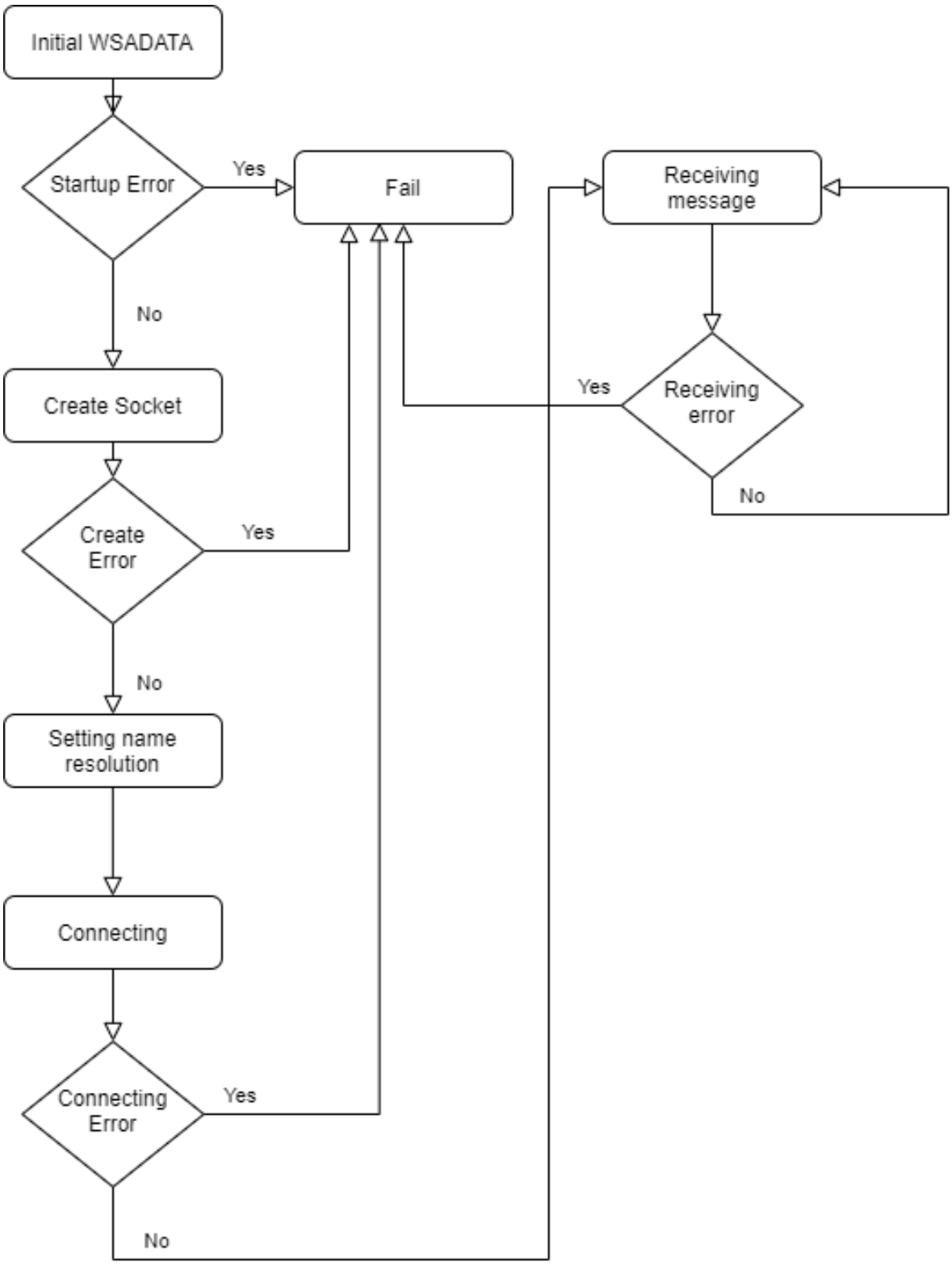

圖 3 Client Reader flow

程式解說

Server

如圖一所示，首先必須將 Windows WSDATA 物件初始化，接著將 WSDAT 作為參數建立一 socket 物件(這裡的 socket 物件僅僅作為連線使用，為表示不同此處稱之為 nFd)，建立完成後進入設定環節，在此步驟 programmer 須將 server 資訊，如: 連線協定、IP、Port…等資訊設定完成。設定完成後便進入綁定程序，所謂綁定就是透過上述建立完成之 server 設定使 socket 物件套用此設定。接著將 socket 設定成監聽模式，如此一來每當有使用者與 server 端建立連線，server 端在該模式下即可透過 accept()接受使用者之連線並進行相對作業。

上述流程為單一作業之 server，稱之為阻塞式 socket。由於本次實驗目標為一多人聊天室，而阻塞式 socket 本身在同一時間只能接受一位使用者之連線，因此本次實驗使用非阻塞式 socket 作為 server 架構。所謂非阻塞式 socket 就是將 socket 透過陣列宣告建立一定數量之 socket，此處宣告之 socket 陣列與上 nFd 不同，在此處宣告之 socket 是用來記錄存放每位使用者之 socket 而 nFd 是為了暫時存放使用者連線之 socket。而對於讀取接收使用者訊息，非阻塞式 socket 與阻塞式 socket 不同在於，非阻塞式 socket 運用 select 作為訪問是否有訊息傳入之行為，而 select 會將使用者 socket 存放在於 set 上，set 又可細分為讀取 set 以及寫入 set ， 因此在以下源碼中可看見，server 在運行迴圈內會不斷呼叫 select 函數，用以偵測有無訊息傳入。假設此時有一使用者傳遞訊息指 server，server 可透過 select 得知有訊息傳入，即可透過原先宣告之 socket 陣列中提取當前使用者 socket 並透過 recv 函數接收訊息，並對陣列內所有 socket 進行廣播使所有使用者皆可收到訊息。

Client

不論是非阻塞式 socket 或是阻塞是 socket，client 通常不會有太多變化。在此次實驗中，因為每當使用者等待輸入時，console 畫面會因 scanf ( ) 卡住而無法刷新聊天訊息，為此本次實驗為使用者提供兩支程式分別為，傳輸用及讀取用，程式流程如圖二及圖三。Client 端建立連線前的初始化作業與 server 端相同，直至創建 socket 物件後，client 使用 connect 函數與 server 要求建立連線(與 server 不同，在 server 中使用 binding 進行 socket 綁定)。 在讀取用程式中 client 會進入迴圈中不斷透過 recv 函數接收 server 傳送之資訊，而傳輸用程式則是不段透過 scanf 函數讀取使用者訊息輸入並將該輸入字串透過 send 函數傳送至 server。

結論

透過此次實驗成功做出一非阻塞式 socket 多人連線聊天系統，但由於 client 等待輸入過程中會因為輸入函數造成畫面卡住無法更新聊天紀錄，因此本次實驗提供兩支程式供使用者使用此為方法一，方法二則為利用圖形化介面將 client 端接收及傳送兩種動作分別寫作成 API 形式，如此一來使用者介面中即可透過按鈕或是圖形化介面將兩種動作平行化處理。相關 Demo 及程式碼請連結至以下相關連結。

Demo 影片: https://www.youtube.com/watch?v=DpEPXVtFCgs

程式原始碼: https://github.com/CodeMachine0121/Non-blocking-socket-multi-chatRoom

參考文獻

Window Socket 筆記: http://www.cchsu.com/arthur/prg_bg5/winsock.htm

WinSock tutorial : https://www.binarytides.com/winsock-socket-programming-tutorial/

Code

Server

```
#include <stdio.h>
#include<winsock2.h>

#pragma comment(lib, "ws2_32") // windows socket l
ibrary
#define WSA_VERSION MAKEWORD(2, 2)

#define SOCK_SIZE 20


struct client_informations{
    char *host;
    int port;
};


int SocketWrite(SOCKET sock, fd_set writerset, cha
r *message, int len){
    if(FD_ISSET(sock, &writerset)){
         // cut fraction of buffer
        //char bufsend[] = "say: Hello from server
\n",);
        // 會一直跑回圈 要看
        printf("msg: %s\n", message);
        send(sock, message, len,0);
        return 1;
    }message="";
    return 0;
```

```
}
int for_SocketWrite(SOCKET socks[], fd_set writers
et, char *message, int len,int times){

    for (int i = 0; i < times; i++)
    {
        SocketWrite(socks[i+1],writerset,message,l
en);
        printf("Send message to sock[%d]\n", i+1);
    }

    return 0;
}



int main(void){
    char *message="";
    WSADATA wsa={0};


    // initialize
    printf("\nInitializing.....\n");
    if(WSAStartup(MAKEWORD(2,2),&wsa) != 0){
        printf("Failed. Error Code: %d\n", WSAGetL
astError());
        return 1;
    }

    printf("Initialized\n");

// creating socket
    int nFd = socket(AF_INET, SOCK_STREAM, IPPROTO
_TCP);
```

```
    if(nFd==INVALID_SOCKET){
        printf("Error creating socket, ec:%d\n", W
SAGetLastError());
        return -1;
    }



// name resolution
    struct sockaddr_in oAddr;
    oAddr.sin_family=AF_INET;
    oAddr.sin_addr.s_addr = INADDR_ANY;
    oAddr.sin_port= htons(8881);


// binding
    if(bind(nFd, (struct sockaddr*)&oAddr, sizeof(
oAddr)) == SOCKET_ERROR){
        printf("Error binding sokcet, ec:%d\n", WS
AGetLastError());
        closesocket(nFd);
        return -1;
    }

// listening
    if(listen(nFd, SOCK_SIZE) == SOCKET_ERROR){
        printf("Error listening, ec: %d\n", WSAGet
LastError());
        closesocket(nFd);
        return -1;
    }

// select
```

```
    u_long  unblock=1;
    if(ioctlsocket(nFd, FIONBIO, &unblock) == SOCK
ET_ERROR){
        printf("ioctlsocket(acp) erroe, ec:%d\n",
WSAGetLastError());
        return -1;
    }

    fd_set ORSet;
    fd_set OWSet;
    FD_ZERO(&ORSet);
    FD_ZERO(&OWSet);
    FD_SET(nFd,&ORSet);
    FD_SET(nFd,&OWSet);

    SOCKET socks[64];
    struct client_informations clis[64];

    socks[0]=nFd;
    clis[0].host="127.0.0.1";
    clis[0].port=80;
    int TotalSockets=1;

    int c,j=0;

    while(1){
        fd_set readerset = ORSet;
        fd_set writerset = OWSet;


        struct timeval tv;
        tv.tv_sec = 1;
```

```
        tv.tv_usec = 0;

        int res = select(0,&readerset,&writerset,N
ULL,&tv);
        if(res == SOCKET_ERROR){
            printf("select error, ec:%d\n", WSAGet
LastError());
            break;
        }

        if(res==0){

            continue; // no connect
        }

        int temTotalSockets = TotalSockets;
         // receive message
        int i;

        for(i=0;i<TotalSockets;i++){

            if(FD_ISSET(socks[i], &readerset)){
                // listensocket 可讀


                if(socks[i]==nFd){ // 第一次連線
                    struct sockaddr_in  client;
                    SOCKET acp = accept(nFd, (stru
ct sockaddr *)&client, &c);

                    if(acp==INVALID_SOCKET){
                        printf("accept error, ec:%
d\n", WSAGetLastError());
```

```
                        break;
                    }

                    if(ioctlsocket(acp, FIONBIO, &
unblock) == SOCKET_ERROR){
                        printf("ioctlsocket(acp) e
rroe, ec:%d\n", WSAGetLastError());
                        break;
                    }

                    FD_SET(acp, &ORSet);
                    FD_SET(acp, &OWSet);
                    //printf("temTotalSockets: %d\
n", temTotalSockets);
                    socks[temTotalSockets++]=acp;


                    // get client address informat
ion and save it into struct client_informations
                    //char* addr_cli = ;
                    clis[j].host = inet_ntoa(clien
t.sin_addr);
                    clis[j].port = client.sin_port
;
                    printf("one connection %s:%d b
y socket[%d] create clis[%d]\n", clis[j].host, cli
s[j].port,i,j);
                    j++;
                }else{

                    // recv message
                    char buf[1024];
                    int len = recv(socks[i], buf,
1024, 0);
```

```
                    if(len==0){

                        printf("No Sleep\n");
                        //break;
                    }else if( len==SOCKET_ERROR){;
                        //printf("Recv error, ec:%
d\n", WSAGetLastError());
                        //break;
                    }else{
                        //char outbuf[len+1];
                        //memcpy(outbuf,buf,len);
                        //outbuf[len] = 0;
                        // '[',']',': '=4
                        int len = strlen(clis[i-
1].host) + strlen(buf) + 1 + 4;
                        char msg[len];
                        memset(msg, '\0', len);

                        strcat(msg,"[");
                        strcat(msg,clis[i-
1].host);
                        strcat(msg,"]: ");
                        strcat(msg,buf);

                        message = msg;
                        printf("recv: '%s' by sock
s[%d]\n",msg, i);
                        printf("from %s:%d  \n",cl
is[i-1].host, clis[i-1].port);
```

```
                        for_SocketWrite(socks, wri
terset, message, strlen(message),j);
                        //free(outbuf);
                    }
                }

            }


        }

        TotalSockets=temTotalSockets;
    }



    closesocket(nFd);
    WSACleanup();
    return 0;
}
```

Client Writer

```
#include <stdio.h>
#include<winsock2.h>

#pragma commnet(lib, "ws2_32.lib") // windows sock
et library

int main(void){
    WSADATA wsa;

    SOCKET s;
    int rc;
    struct sockaddr_in server;

    printf("\nWinsock initializing....\n");
    if(WSAStartup(MAKEWORD(2,2),&wsa)!=0){
        printf("Failes. Error Code: %d\n",WSAGetLa
stError());
        return 1;
    }

    printf("Initialized\n");

    // Create socket
    if((s=socket(AF_INET, SOCK_STREAM, 0))==INVALI
D_SOCKET){
        printf("Could not create socket: %d\n", WS
AGetLastError());
    }

    printf("Socket created\n");
```

```
    //設定連線
    server.sin_addr.s_addr = inet_addr("127.0.0.1"
);
    server.sin_family = AF_INET;
    server.sin_port = htons(8881);

    // Connect to remote server
    rc = connect(s,(struct sockaddr *)&server, siz
eof(server));
    if(rc<0){
        printf("connect error\n");
        return 1;
    }

    printf("Connected\n");
    while(1){


        char *str;
        str = (char *)malloc(sizeof(char)*1024);
        printf("> ");
        scanf("%s", str);

        if(send(s, str, 1024, 0)<=0){
            printf("send error\n");
            return 1;
        }
        printf("\"%s\" sent\n",str);


    }
```

```
    closesocket(s);
    WSACleanup();
    return 0;
}
```

Client Reader

```
#include <stdio.h>
#include<winsock2.h>

#pragma commnet(lib, "ws2_32.lib") // windows sock
et library

int main(void){
    WSADATA wsa;

    SOCKET s;
    int rc;
    struct sockaddr_in server;

    printf("\nWinsock initializing....\n");
    if(WSAStartup(MAKEWORD(2,2),&wsa)!=0){
        printf("Failes. Error Code: %d\n",WSAGetLa
stError());
        return 1;
    }

    printf("Initialized\n");

    // Create socket
    if((s=socket(AF_INET, SOCK_STREAM, 0))==INVALI
D_SOCKET){
        printf("Could not create socket: %d\n", WS
AGetLastError());
    }

    printf("Socket created\n");
```

```
    //設定連線
    server.sin_addr.s_addr = inet_addr("127.0.0.1"
);
    server.sin_family = AF_INET;
    server.sin_port = htons(8881);

    // Connect to remote server
    rc = connect(s,(struct sockaddr *)&server, siz
eof(server));
    if(rc<0){
        printf("connect error\n");
        return 1;
    }

    printf("Connected\n");
    while(1){



         // message to send

        char *server_reply; // message to receive
        server_reply = (char *) malloc(sizeof(char
)*1024);
        if( (rc = recv(s,server_reply, 1024, 0)) =
= SOCKET_ERROR){
            printf("recv error\n");
        }
        //server_reply[1024] = ' ';
        printf(">> %s\n", server_reply);
       // free(server_reply);

    }
```

```
    closesocket(s);
    WSACleanup();
    return 0;
}
```